10. Functies, klassen en libraries

Python functies

Een functie is een stuk programmacode dat wordt uitgevoerd wanneer we het oproepen. Python kent een ingebouwde functies. Een voorbeeld is functie print die we al veel gebruikt hebben. Andere voorbeelden zijn de conversiefuncties, die converteren een getal van één type naar een ander. Veel functies zijn in libraries gedefinieerd, je kunt ze oproepen en in eigen software gebruiken.

int(x)	Converteert x naar een geheel getal
float(x)	Converteert x naar een kommagetal
str(x)	Maakt een string van getal x.

Functie str() kent ook de opmaak die we gezien hebben bij functie print(). Voorbeeld:

```
p = 3.141592653589
t = str("%.2f" %p)
Het resultaat is string "3.14"'
```

Voorbeelden van stringfuncties zijn:

len(*string*)	Geeft de lengte van *string*
upper(*string*)	Zet *string* in hoofdletters
lower(*string*)	Zet *string* in kleine letters
isdecimal(*string*)	Gaat na of *string* volledig uit cijfers bestaat

Voorbeelden van listfuncties zijn:

len(*list*)	Geeft de lengte van *list*
max(*list*)	Geeft het grootste element van *list*
min(*list*)	Geeft het kleinste element van *list*
list.reverse()	Geeft *list* in omgekeerde volgorde
list.sort()	Sorteert *list*

Zelf gedefinieerde functies

We hebben in vorige hoofdstukken alleen kleine programma's gebruikt. Die bestaan uit één stuk programmacode. Dat werkt dit perfect maar voor grotere programma's verlies je op die manier nogal snel het overzicht. Een programma wordt veel leesbaarder als we het opsplitsen in verschillende modules die specifieke taken uitvoeren. We noemen zo'n module een functie of een methode. Een **functie** is een

groep opdrachten die samen één taak uitvoeren. Het hoofdprogramma blijft het startpunt, van daar uit worden functies opgeroepen. Het gebruik van functies is ook handig als we een zelfde berekening op verschillende plaatsen van het programma gebruiken.

voorbeeld

```
def discriminant (a, b, c):
    "discriminant van een vierkantsvergelijking"
    d = b*b-4*a*c
    return d
```

De algemene vorm van een functiedefinitie is

def *functienaam* (*parameters*):
    "functiestring"
    berekeningen
    return *resultaten*

De functiestring is een string die aangeeft waarvoor we de functie gebruiken. Je mag die weglaten. De laatste regel is return, eventueel gevolgd door resultaten die je wil terugsturen.

Je roept de functie op door

A = discriminant (1, 2, 3)
B = discriminant (x, y, z)

Met het return-statement kan je meerdere resultaten terugsturen. In het voorbeeld hieronder maken we een functie die tegelijk de sinus en de cosinus van een getal berekenen. Die twee waarden staan in de return-regel.

```
import math
def sincos (x):
    si = math.sin(x)
    co = math.cos(x)
    return si, co

pi  = 3.14159265
s, c = sincos(pi)
print ("sin(%f) = %f" %(pi, s))
print ("cos(%f) = %f" %(pi, c))
```

Recursiviteit

Een **recursieve functie** roept zichzelf aan.

Voorbeeld:

De wiskunde definieert faculteiten als producten:

5! = 1.2.3.4.5 = 120 (5 faculteit)
2! = 1.2 = 2
1! = 1

De wisunde definieert 0! = 1

Meer algemeen is

n! = 1.2. … .n en 0! = 1

Je ziet dat 5! = 5x4! of meer algemeen

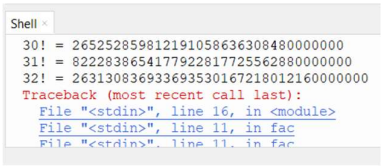
n! = n.(n-1)!

Die laatste vorm gebruiken in het programma:

```
#------------------------------#
#  faculteit.py                #
#                              #
#  27 februari 2021            #
#  Dirk Ghysels                #
#------------------------------#
def fac (n):
    if (n==0):
        f = 1
    else:
        f = n*fac(n-1)
    return f

n = 1
while True:
    fa = fac(n)
    print ("%u! = %u" %(n, fa))
    n += 1
```

Het programma berekent 5! m.b.v. 4!, 4! Met 3! en zo verder. Het aantal maal dat de functie zichzelf achtereenvolgens oproept is de recursiediepte en die is beperkt. Het programma stopt met een foutmelding als de maximale recursiediepte is bereikt. Dat is, verrassend genoeg, afhankelijk van de controller. MicroPython op een RP2040 blijft faculteiten berekenen en afdrukken tot 32!, de maximale recursiediepte is bereikt. MicroPython op een esp32 loopt vast na 50!.Een ESP32-C2 houdt het vol tot 25!. De ESP8266 loopt vast na 20! De SAMD21 met een beperkte MicroPython implementatie doet het tot 43!



Figuur 39: de maximale recursiediepte is bereikt

globale en lokale variabelen

Variabelen die we in een functie definiëren zijn lokaal. Ze zijn alleen binnen de functie gekend. Omgekeerd, de waarde van een variabele buiten een functie is niet gekend binnen de functie.

```
x = 5

def func ():
    x = 14
   …
    return k

y = func()
print (x)
```

De eerste x is gedefinieerd buiten de functiedefinitie. Die waarde is niet gekend binnen de functie. Daar definiëren we een andere variabele met dezelfde naam x met waarde 14. Deze variabele is een **lokale variabele**. De laatste regel drukt x af, dat is waarde 5

Een **globale variabele** is gekend buiten én binnen een functie. We gebruiken daarvoor sleutelwoord **global**:

```
x = 5

def func ():
    global x
    x = 33
    …
    return k

y = func()
print (x)
```

Binnen de functie kunnen we x=5 gebruiken en kunnen we de waarde veranderen. Op het einde wordt 33 afgedrukt.

Klassen en objecten

Python is **object-georiënteerd**. Het maken van objecten is vrij eenvoudig. In dit hoofdstuk leren we wat klassen en objecten zijn en hoe we die met Python kunnen gebruiken. Maar eerst bekijken we enkele definities.

- Een **klasse** is prototype voor een object en definieert attributen die het object of de klasse bepalen. De attributen zijn variabelen en methodes.
- Een **klassevariabele** is een variabele die gedeeld wordt door alle instanties van de klasse.
- Een **instantievariabele** is is gedefinieerd binnen een klasse-functie en behoort tot één instantie van de klasse.
- Een **instantie** is een vertegenwoordiger van de klasse.
- Een **object** is een unieke gegevensstructuur die gedefinieerd is door de klasse.
- Een **methode** is een functie die binnen een klassedefinitie gedefinieerd is.
- **Overerving** is de overdracht van kenmerken van een klasse naar een andere klasse die er is van afgeleid.

Je zou een klasse kunnen vergelijken met een type en een object met een variabele met de klasse als type, maar klassen zijn veel meer dan dat.

Een klasse definiëren

Met opdracht **class** maken we een nieuwe klasse. De algemene vorm is:

class *KlasseNaam*:
    '*documentatiestring*'
    klasse-attributen

De *documentatiestring* is toegankelijk via variabele *KlasseNaam.__doc__*. Let op, er staan twee underscore-tekens voor en na doc.

Voorbeeld:

We maken een klasse voor de studenten van een school:

```
class Student:
        'klasse voor de leerlingen van school KTA'
        studAantal = 0
        def __init__(self, naam, leerjaar):
                self.naam = naam
                self.leerjaar = leerjaar

                Student.studAantal += 1
        def toonaantal(self):
                print "aantal studenten: %d" %Student.studentAantal

        def toonStudent(self):
                print ("Naam: %s" %self.naam)
                print ("Leerjaar: %s" %self.leerjaar)
```

- Variabele *studAantal* is een klassevariabele en is gedeeld door alle vertegenwoordigers van de klasse. Hij is toegankelijk als *Student.studAantal*.
- Methode __init__() is de klasseconstructor. Let op, hier ook dubbele underscores voor en na. Je gebruikt hem om nieuwe vertegenwoordigers aan te maken. De eerste parameter is altijd *self*, maar die neem je niet op bij de aanroep.
- We declareren de ander methodes als gewone functies, behalve dat *self* telkens het eerste argument is.

Hoe gebruiken we de klasse?

- Declaratie van vertegenwoordigers:
  - student1 = Student("Marcie Palmer", "3de leerjaar")
  - student2 = Student("Ilse Maes", "5de leerjaar")
    hier gebruik je de parameters uit *__init__*
- Het totaal aantal studenten:
  - print ("aantal studenten : ", Student.studAantal)
- Eigenschappen van een student:
  - Student1.toonStudent()

Libraries

Een library of bibliotheek is een bestand met functies en/of klassen die we in andere programma's kunnen gebruiken. Een functiebibliotheek is een library waarin alleen functies zijn gedefinieerd. Voor je de library gebruikt moet je ze in jouw project opnemen met **import**. Dat zien we verder in de voorbeelden. Voor veel toepassingen gebruiken we externe libraries. Die downloaden we als een bestand met extensie .py. Dat bestand moet je kopiëren naar de controller: open het in Thonny en bewaar het dan (Save as…) naar de Raspberry Pi Pico. Voor veel toepassingen vinden we libraries op Internet. Vooral op Github.com vind je veel voorbeelden van MicroPython projecten. De RP2040 is nieuw, veel toepassingen zijn gemaakt voor de andere platforms: ESP8266, ESP32, of het pyboard. Meestal kan je die direct gebruiken voor ons systeem.

Modules: voorgedefinieerde libraries:

In het MicroPython system zijn een heleboel libraries ingebouwd. Sommige libraries zijn hardware-afhankelijk: een RP2040 werkt intern anders dan een ESP32. Daarom verschillen die libraries naargelang de hardware waarop ze moeten werken, sommige libraries zijn ook niet uitgewerkt voor alle platformen. Een lijst van de modules van jouw systeem vind je met

```
help('modules')
```

Figuur 40: modules van de ESP8266

Python standaard libraries

Deze behoren tot de Python standaard en zijn aangepast aan het MicroPython systeem. Enkele voorbeelden:

Module array

Met module **array** kan je een array (tabel) met getallen maken.

Module math

math is een library met wiskundige functies. Voor je een functie uit die module gebruikt, moet je aangeven dat je die module gaat gebruiken:

```
import math
pi = 3.14159265
x = math.sin(pi)
print (x)
```

De eerste regel geeft aan dat we module math gaan gebruiken. Die wordt geïmporteerd in het programma. In de derde regel gebruiken we de sinusfunctie. Die is onderdeel van math, daarom zetten we prefix math. voor de functieaanroep.

Andere wiskundige functies zijn

math.abs(x)	Absolute waarde
math.abs(x)	Converteert x naar een kommagetal
math.exp(x)	e^x
math.log(x)	Natuurlijke logaritme
math.log10(x)	Logaritme, grondtal 10
math.log2(x)	Logaritme, grondtal 2
math.pow(x,y)	x^y
math.sqrt(x)	Vierkantswortel
math.floor(x)	Grootste geheel, $\leq$ x
math.ceil(x)	Kleinste geheel, $\geq$ x

math.gamma(x)	Gamma functie
math.isfinite(x)	True als x≠
math.isinf(x)	True als x=∞
math.isnan(x)	True als x geen getal is

Goniometrische functies vind je ook in math:

math.sin(x)	Sinus, x in radialen
math.cos(x)	Cosinus
math.tan(x)	Tangens
math.asin(x)	Arcsin, resultaat in radialen
math.acos(x,y)	Arccos
math.atan(x)	Arctangens
math.degrees(x)	Radialen naar graden
math.radians(x)	Graden naar radialen

Math heeft twee constanten:

math.e	2.7182817
math.pi	3.1415927

Module cmath

De functies uit math kan je niet gebruiken bij complexe getallen, daarvoor gebruiken we module cmath. Die werkt op dezelfde manier en met dezelfde functies. De meeste functies uit math hebben een complexe tegenhanger in cmath. Enkele voorbeelden:

```
import cmath
y = cmath.sin(3+4j)
print ("y = ", y)

z = cmath.sqrt(y)
print("z = ", z)
```

```
Shell
>>> %Run -c $EDITOR_CONTENT
  y =  (3.853739-27.01682j)
  z =  (3.946139-3.423196j)

>>>
>>>
```

Figuur 41: module *cmath*

Niet elke math-functie heeft een complexe tegenhanger: asin, acos en atan bestaan niet in cmath.

Module os

Deze module geeft toegang tot elementaire operating system functies, zie een volgend hoofdstuk

Module sys

Deze module geeft functies en constanten i.v.m. de gebruikte versie van MicroPython. Enkele voorbeelden:

```
>>> import sys
>>> sys.maxsize
2147483647
>>> sys.platform
'esp8266'
>>> sys.implementation
(name='micropython', version=(1, 17, 0), mpy=9733)
```

Figuur 42: module sys/usys

- sys.maxsize geeft het grootste geheel getal dat deze versie kan weergeven
- sys.platform geeft aan dat we de esp8266-gebruiken. Dit kan nuttig zijn als je een programma maakt voor verschillende platformen.
- sys.implementation zegt dat we MicroPython versie 1.17 gebruiken.

Module Random

Met random brengen we een stuk willekeur in ons programma: een willekeurig getal of een willekeurig element uit een list.

Een willekeurig geheel getal vorm je met **functie randint():**

```
import random
y = random.randint (1, 6)
geeft een willekeurig getal van 1 tot en met 6
```

randrange(n) doet ongeveer hetzelfde:

```
y = random.randrange(6)
geeft een willekeurig getal van 1 tot 6, 6 niet inbegrepen
```

Een willekeurg float getal tussen 0 en 1 krijg je met **random():**

```
y = 35*random.random ()
geeft een willekeurig getal tussen 0 en 35
```

Met **choice()** kiezen we uit een list:

```
namen = [“Jan”, “Piet”, “Joris”, “Korneel”]
y = random.choice (namen)
```
geeft een willekeurige naam uit *namen*

MicroPython specifieke libraries

Deze zijn specifiek tot de MicroPython. Enkele voorbeelden:

Module bluetooth

Met deze library kan je een bluetooth systeem opbouwen.

Module machine

Deze library bevat functies en klassen i.v.m. de hardware. Enkele voorbeelden van functies:

```
>>> print (machine.freq())
125000000
>>> machine.reset()
>>> machine.unique_id()
b'\xe6`8\xb7\x13\x92=/'
```

De eerste geeft de kloksnelheid van de processor weer: 125000000 Hz of 125 MHz. De tweede reset de machine. De derde geeft het unieke identificatienummer dat in de RP2040 is opgeslagen.

Een voorbeeld van een klasse in **machine** is **klasse Pin**. Pin controleert de GPIO’s. We importeren Pin met

```
From machine import Pin
```

We maken een vertegenwoordiger van de klasse met

```
pinA = Pin(id, mode, pull, drive, alt)
```

De eerste twee parameters zijn verplicht, de andere mag je weglaten.

- **id** is het nummer van de GPIO
- **mode** kan volgende waarden aannemen: Pin.IN geeft aan dat we de GPIO als input gebruiken, Pin.OUT gebruiken we voor een outputpin.
- *De andere parameters komen in een volgend hoofdstuk aan bod.*

Enkele methodes van Pin zijn

Pin.value(x) stelt de waarde van een output in, x heeft waarde 0 of 1.
Pin.toggle() verandert de output van 1 naar 0 of omgekeerd.
methode toggle() werkt niet bij een esp32

Andere klassen van library machine komen later aan bod.

Module network
Deze library configureert het netwerk, zie ook een volgend hoofdstuk.

Module neopixel
Deze library controleert neopixel LED's WS2812, zie ook een volgend hoofdstuk.

Module ssd1306
Deze library controleert kleine beeldschermen met een SSD1306 aansturing, zie ook een volgend hoofdstuk.

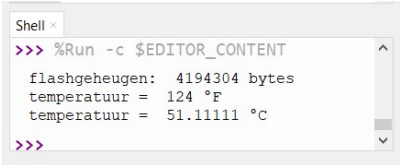
Poort-specifieke libraries

Deze werken voor één specifieke poort (een implementatie van MicroPython voor een controller of een controllerbordje) Een voorbeeld:

Modules esp en esp32 *voor ESP-controller*

esp.flash_size() geeft de grootte van het flash geheugen
esp32.raw_temperature() leest de interne temperatuursensor, in fahrenheit (*voor esp32, niet voor esp32-c3*)

```
import esp32,esp
print("flashgeheugen: ",esp.flash_size(), "bytes" )
tF = esp32.raw_temperature()
print("temperatuur = ", tF, "°F")
tC = (tF-32.0)/1.8
print("temperatuur = ", tC, "°C")
```



Figuur 43: modules esp en esp32

Methode raw_temperature() meet de temperatuur van de chip. Die wijkt soms sterk af van de omgevingstemperatuur.

Uitgebreide libraries

Van veel modules bestaat er een uitgebreide versie, die we aanduiden met letter u, zoals **time** en **utime**. Je mag ze allebei gebruiken tenzij je de specifieke elementen uit de uitgebreide nodig hebt. Blijf wel consequent in een programma. Volgende constructie geeft een **foutmelding**:

```
import utime
```

```
time.sleep(1)
```